Modernizing with AI: Using Intelligent Automation to Upgrade Legacy Code

Key Takeaways

- **AI-powered application modernization is augmentation, not autopilot.** The teams winning at this in 2026 are pairing generative AI with deterministic static analysis and keeping humans firmly in the loop.
- **The biggest unlock isn't code translation — it's understanding.** AI is finally cracking the "nobody knows what this system does anymore" problem that's blocked legacy modernization for two decades.
- **COBOL isn't disappearing — it's being surrounded.** Roughly 70% of global banking transactions still run on mainframes, and around 84% of IBM Z customers still run COBOL. AI tools like watsonx Code Assistant for Z translate selectively rather than rewriting wholesale.
- **The hallucination problem is smaller for code than for prose.** Code either compiles and passes tests or it doesn't. That's a more honest feedback loop than AI gets in most other domains.

- **The real risk isn't AI getting things wrong — it's humans rubber-stamping AI output.** A Stanford study found developers using AI assistants produced less secure code, mostly because they reviewed less carefully. Governance matters more than the model.
- **Gartner expects GenAI to cut modernization costs by up to 70% by 2027.** That number is doing a lot of work in 2026 budget conversations — and for once, it's plausible.

For years, modernizing legacy code meant one of two bad options. You could rewrite the whole thing — expensive, risky, and usually abandoned halfway through when the budget ran out. Or you could keep patching, keep paying COBOL contractors a small fortune, and hope nothing broke before retirement. Both options aged badly.

What changed in 2026 isn't that AI got smart enough to "rewrite your monolith." It didn't, and anyone telling you otherwise is selling something. What changed is that AI got good enough at the *boring* parts of modernization — reading code nobody's looked at in fifteen years, mapping dependencies, generating tests that should have existed all along, translating idioms between languages — that the economics of the whole project shifted.

This is a practical guide to what's actually working, what's still hype, and where the real risks sit.

What Is AI-Powered Application Modernization?

AI-powered application modernization is the use of generative AI, intelligent automation, and agentic systems to accelerate and de-risk the process of upgrading legacy code. That's the dry definition. Here's the real one: it's everything that used to take a senior engineer six months of code archaeology, done in days, with the engineer redirected to the parts that actually require judgment.

What it isn't: a magic rewrite button. Gartner's category for this — AI-Augmented Code Modernization Tools — is deliberately named *augmented*, not autonomous. The pattern that's working in production combines three layers:

A **deterministic layer** that statically analyzes code, maps dependencies, and produces verifiable, traceable insights. This is what catches the things you can't afford to get wrong.

A **generative layer** built on large language models tuned for code (IBM's 20-billion parameter Granite model is the canonical example, trained on 1.6 trillion code tokens and fine-tuned on COBOL-Java pairs). This handles the creative work — refactoring suggestions, language translation, documentation generation, test scaffolding.

A **human layer** that reviews, approves, and ships. This is the layer that gets cut by vendors selling "fully autonomous" stories, and it's also the layer that prevents disasters.

When all three are in place, modernization stops being a doomed multi-year program and starts looking like a managed process. McKinsey, Deloitte, and Thoughtworks have all landed on roughly the same conclusion: AI compresses timelines and reduces toil dramatically, but only with strong guardrails and expert oversight. The teams skipping the guardrails are also the teams writing case studies titled "What We Learned From Our Failed AI Modernization."

How AI Is Transforming Legacy Code Modernization

The shift isn't in any single capability — it's that the entire modernization workflow now has AI working underneath it. Here's where it's making the most difference, in roughly the order you'd encounter them.

Discovery and code comprehension. This is where AI is having its biggest impact, and it's the least sexy. Most legacy systems have lost the people who built them. Documentation is missing, outdated, or aspirational. AI can now read millions of lines of unfamiliar code and produce architectural diagrams, dependency maps, and plain-English explanations of what each module actually does. Six months of consulting fees, compressed into a few automated runs.

Documentation generation. Following from comprehension: AI writes the documentation the original team never did. Inline comments, architecture notes, runbook content. Not because the docs are perfect, but because *something* readable is infinitely better than nothing, and human engineers can edit faster than they can write from scratch.

Automated refactoring. AI suggests safer changes — extracting methods from oversized functions, renaming variables for clarity, replacing deprecated API calls with modern

equivalents. The pattern that works: AI proposes, humans review, automated tests verify. The pattern that fails: AI proposes, humans approve in bulk, tests get added later (they don't).

Language translation. The headline use case. IBM's watsonx Code Assistant for Z is the canonical example: it translates COBOL business services to Java, doesn't just transliterate syntax, and uses an LLM specifically fine-tuned on enterprise COBOL-Java pairs. The output is object-oriented Java that interoperates with the rest of the COBOL application via CICS, IMS, and DB2. The same playbook is being applied to other migrations — classic ASP to .NET, older Spring to Spring Boot, PHP 5 to modern PHP or Node.

Test generation. Legacy systems almost always lack meaningful test coverage. AI generates unit and integration tests against existing behaviour, which is the foundation for everything else. You can't safely refactor what you can't verify, and writing tests by hand for an undocumented monolith is what kept these projects underwater for years.

Continuous monitoring and root-cause analysis. Post-deployment, AI handles anomaly detection and increasingly auto-remediation. Augment Code's data shows AI testing frameworks identifying 87% of critical defects within the first 30% of test execution time, and automated root-cause analysis cutting diagnosis time by 45-55%. The interesting consequence: the entire modernization economics changes when post-deployment cost drops by half.

Agentic workflows. This is what's new in 2026. Earlier AI assistants responded to single prompts. Agentic systems orchestrate multi-step workflows, maintain context across multiple repositories, and run scheduled assessments on their own. A single agent might handle "review every service in this monorepo for deprecated dependencies, propose upgrades, generate tests, open PRs, and tag the team lead for review." That's a different shape of work entirely.

Challenges of Maintaining Legacy Applications

Before we get to the upside, it's worth being honest about why this matters. The case for AI-driven modernization rests on how genuinely painful legacy maintenance has become.

The skills are gone. COBOL is the obvious example. It's over 65 years old. The people who wrote it have largely retired. The remaining COBOL developers can name their price, and they do. The same talent crunch is now hitting older Java versions, classic ASP, on-prem Oracle Forms, and the long tail of in-house frameworks every enterprise has somewhere.

Documentation is fiction or absent. Legacy systems were built when "the code is the documentation" was a respectable engineering position. It wasn't, but here we are.

Architecture diagrams are out of date if they exist. Tribal knowledge walked out the door with the last round of layoffs.

The maintenance tax is enormous. U.S. enterprises were already spending around $85 billion a year just keeping outdated systems alive. Gartner estimated companies would burn 40% of their IT budgets on technical debt by 2025, and that prediction landed accurately. Every dollar funding legacy life support is a dollar not funding AI, automation, or any of the things that move the business forward.

The systems block AI initiatives directly. This is the new pressure point. Legacy architectures weren't designed to expose APIs, stream events, or expose clean data interfaces — all of which modern AI workloads need. You can't bolt a recommendation engine onto a monolith that processes everything overnight in batch. Cloudflare's 2026 App Innovation Report found companies that modernized first were three times more likely to see real ROI from their AI investments.

Security debt compounds. Old authentication patterns, unencrypted internal traffic, ancient cryptographic libraries, hardcoded credentials, dependencies that haven't been patched in years. Attackers know these patterns better than the people maintaining the systems do.

Change becomes terrifying. When everyone who understood the system is gone, every change is risky, so changes get rarer, which means knowledge atrophies further. It's a feedback loop, and it accelerates.

This is what **AI-driven modernization** is actually solving. The technology is impressive, but the reason it matters is that the alternative — keeping these systems alive manually — has become economically and operationally untenable.

Benefits of
Intelligent Automation
for Legacy Systems
INFOSWIFT
Robust. Secure. Solutions.
LEGACY SYSTEM
AI
INTELLIGENT AUTOMATION
Future-Ready
Prepare your business for future technologies and innovations.
Increased Efficiency
Automate manual processes and accelerate application modernisation.
Reduced Costs
Lower maintenance costs and minimize operational expenses.
Improved Accuracy
Minimize human errors and ensure consistent code quality.
Improved Time to Market
Modernize legacy applications faster and deliver value more quickly.
Stronger Security
Identify vulnerabilities and ensure compliance with modern standards.
getinfo@infoswift.com
TALK TO OUR EXPERT
Discover how AI-powered automation can modernize your legacy systems, reduce costs, and drive lasting business impact.
TALK TO OUR EXPERT

Benefits of Intelligent Automation for Legacy Systems

A list of vendor talking points isn't useful. Here's what the benefit actually looks like when programs work.

Timeline compression. Tasks that took quarters take weeks. Discovery and dependency mapping that used to require a multi-person team for months now runs as an automated pass. Gartner's projection that GenAI tooling could cut modernization costs by up to 70% by 2027 sounded ambitious when first published. By mid-2026, it doesn't.

Risk reduction at every stage. The combination of AI-generated tests, automated impact analysis, and gradual rollout means refactors land safely. Pre-merge hooks now run automated impact analysis when a PR opens, posting the blast-radius diff as a comment so reviewers know what to scrutinize. Policy gates fail builds when confidence scores drop. Canary deploys monitor anomaly metrics before full rollout. None of this was practical for legacy systems before.

Selective modernization instead of all-or-nothing. This is the strategic shift. With AI-assisted tools, you can surgically extract a single business service from a monolith, transform just that service, validate it, and ship it — without disturbing anything else. IBM calls this "selective and incremental" modernization, which is bland phrasing for what's actually a major change in how these programs get scoped. You no longer need to commit to rebuilding everything to make progress on anything.

Documentation as a free byproduct. AI generates inline comments, architectural notes, and code summaries as it works. By the time the modernization is done, you have documentation you never had before. Onboarding new engineers stops being a six-month process.

Skills gap relief. AI doesn't replace your COBOL specialists, but it dramatically expands what your Java team can safely touch. Junior engineers can contribute meaningfully to mainframe

codebases with AI providing context they'd otherwise lack. This matters because the skills gap isn't a marketing problem — it's a real, growing constraint on every modernization budget.

The compounding benefit nobody talks about. Once you've modernized one service with AI assistance, the next one is easier. Patterns get learned, prompts get refined, internal tooling gets built. The first AI-driven modernization in your org is the hard one. The tenth one ships in a week.

Common Risks and Challenges in
AI-Driven Modernization
While AI accelerates modernization, it's important to navigate potential risks to ensure successful and secure transformation.
TALK TO OUR EXPERT
Get personalized guidance to overcome challenges, mitigate risks, and accelerate your AI-powered modernization journey.
TALK TO OUR EXPERT
Data Quality and Availability
Poor or incomplete data can lead to inaccurate AI insights and suboptimal outcomes.
Security and Privacy Concerns
Legacy systems may have vulnerabilities that AI tools can expose or amplify.
Integration Complexity
Integrating AI with legacy infrastructure can be complex and time-consuming.
High Initial Investment
Upfront costs for tools, talent, and infrastructure can be significant.
Skill Gaps and Talent Shortage
Lack of AI and modernization expertise can slow down progress.
Unclear ROI and Business Value
Without clear goals and metrics, measuring success and ROI can be challenging.
INFOSWIFT
Robust. Secure. Solutions
getinfo@infoswift.com

Common Risks and Challenges in AI-Driven Modernization

This is where I'd diverge from most vendor content. The risks here are real and deserve more honesty than they typically get.

Hallucination, but probably not where you think. The good news: code hallucinations are easier to catch than text hallucinations. Code either compiles and passes tests or it doesn't. IBM's CTO for Z Software, Kyle Charlet, made this point publicly — hallucinated code shows itself fast, unlike hallucinated text, which can read as authoritative while being wrong. The bad news: hallucinations in *understanding* code — wrong dependency maps, missed edge cases, misunderstood business logic — are harder to catch and can quietly poison downstream decisions. The deterministic-plus-generative pattern exists precisely because of this.

The Stanford problem: humans rubber-stamping AI output. A Stanford study found developers using AI code assistants produced *less secure* code than those who didn't. The cause wasn't the AI being unsafe — it was developers reviewing AI output less critically than they'd review their own. This is the central risk of intelligent automation in modernization, and it doesn't get talked about enough. The fix is process, not technology: explicit security review steps, automated scanning gates, peer review of AI-generated code at the same depth as human-written code.

Brittle output from naive line-by-line translation. Early COBOL-to-Java tools produced Java that looked exactly like COBOL written in a different syntax — unreadable and unmaintainable. Modern AI translation aims for *intent-preserving* transformation, mapping COBOL constructs into idiomatic Java patterns. The lesson generalizes: bad AI translation can be worse than no translation, because it locks in legacy patterns under a modern syntax veneer. Vet the output style, not just the output correctness.

The "we automated the boring parts" trap. Yes, AI automates the boring parts. The boring parts are often where the institutional knowledge is buried. If you let AI replace the engineer who would have learned the system by reading the code, you've optimized for speed but created a future skills problem inside your modernized stack. Engineering judgment doesn't develop in teams that never struggle with the code.

Tool sprawl and lock-in. The AI-augmented code modernization market is crowded with overlapping tools — IBM watsonx Code Assistant, Moderne, EvolveWare, BMC AMI DevX, TCS MasterCraft, vFunction, and a dozen others, plus general-purpose copilots used for the same work. Picking one is a meaningful commitment, switching is painful, and most don't interoperate cleanly. Choose for your actual stack, not for marketing.

Data and IP concerns. Your legacy codebase is sensitive intellectual property. Many AI modernization tools were originally cloud-only, sending code to remote LLMs. The 2026 shift is meaningful: IBM's watsonx Code Assistant for Z now runs entirely on-premises, and most serious enterprise tools offer air-gapped or VPC-isolated deployment. If your tool's data path isn't auditable, walk away.

Cost surprises. LLM inference at enterprise scale is not cheap. A modernization program that runs millions of completion requests across a large codebase can rack up bills that surprise everyone. FinOps governance from day one applies here as much as it does to cloud migration.

Over-trusting AI on regulated workloads. In financial services, healthcare, and government, AI-generated code modifications need explicit audit trails, version control of model outputs, and regulatory sign-off. Some industries aren't ready for fully AI-driven changes regardless of how good the model is. Know your compliance environment before the program starts, not after.

A Quick Note on Choosing Tools

I won't do a vendor rundown because it ages badly within months, but the questions worth asking:

Is the tool's output style *idiomatic* in the target language, or does it just translate syntax? Does it support your specific source language and target stack, not just the marketed-up demos? Can it run in your data environment (on-prem, VPC, air-gapped) or only in the vendor cloud? Does it produce auditable trails for every transformation? Does it integrate with your CI/CD, or does it want to be its own delivery system? And — often the deciding question — does it work *with* your existing modernization patterns (Strangler Fig, modular extraction) or does it want you to do everything its way?

Gartner's category for AI-Augmented Code Modernization Tools is a reasonable starting list. Beyond that, run paid pilots on real code. Demo environments lie. Your codebase tells the truth.

How to Get Started Without Overcommitting

If you're considering **AI-driven modernization**, the lowest-risk way in is also the highest-value: start with **discovery and documentation**. Run an AI-powered code comprehension pass on a system nobody fully understands. You'll get architectural diagrams, dependency maps, and plain-language explanations of what the code does. That output alone justifies the pilot cost in most cases, and it gives you the foundation for everything else.

Then layer in **test generation** for the same system. Now you have coverage where you had none. Then **refactoring suggestions**, reviewed and applied gradually. Then **language translation**, if that's relevant. Each step builds on the last, each ships value independently, and you can stop at any point without having wasted the investment.

The teams getting this right in 2026 aren't running massive "AI modernization initiatives." They're embedding AI tooling into normal engineering work, project by project, and letting the capability accumulate.

FAQs

What is AI-powered application modernization? It's using generative AI and intelligent automation to upgrade legacy code — analyzing dependencies, generating documentation, translating languages, and producing tests. Humans still decide what to modernize and verify the output. AI accelerates the work that used to take months of manual code archaeology.

How does intelligent automation help modernize legacy code? It reads the codebase, maps undocumented dependencies, surfaces buried business logic, generates test coverage where none existed, and produces refactored or translated code for human review. The combination of deterministic static analysis with generative AI is what makes this actually safe in production environments.

Why is legacy code modernization important? Old code blocks AI initiatives, drives roughly $85 billion in annual U.S. maintenance spend, creates security vulnerabilities, and starves you of engineers who'll touch it. Companies that modernized first are three times more likely to see real returns from their AI investments today.

What are the benefits of AI-driven software modernization? Faster discovery, automated documentation, test generation for code that never had any, AI-assisted language translation, and dramatically compressed timelines. Gartner expects GenAI to cut modernization costs by up to 70% by 2027 — savings that genuinely change the business case for transformation.

How does AI improve code refactoring? AI suggests safer refactors, replaces deprecated APIs, breaks up oversized functions, and renames variables for clarity — all with regression tests generated against existing behaviour. The risk drops sharply because you're not refactoring blind; you're refactoring with coverage the AI created for you first.

Can AI automate application migration? Partially. AI reliably handles language translation, dependency mapping, and test generation. The architecture decisions, business prioritization, and cutover strategy still belong to humans. Treat AI as an accelerator that compresses months into weeks, not as a replacement for engineering judgment or governance practices.

InfoSwift Application Modernization helps businesses transform legacy applications into scalable, cloud-ready solutions that improve performance, agility, and user experience.
Our modernization services enable organizations to reduce technical debt, accelerate innovation, and maintain competitive advantage through secure, future-ready architectures.
Infoswift delivers end-to-end digital transformation and IT services across the United States. Cloud, AI, QA testing, app modernization & enterprise solutions. Get a free consultation.
Visit: www.infoswift.com/ services/migration/
DM Us: getinfo@infoswift.com
Call: +1 (714) 660-6085